UNITED STATES PATENT APPLICATION

METHOD AND APPARATUS FOR CONTROLLING ACCESS TO SHARED RESOURCES IN AN ENVIRONMENT WITH MULTIPLE LOGICAL PROCESSORS

Inventors:

Jason G. SANDRI
Steven J. TU
Orlando R. DAVILA

Prepared By:

Kenyon & Kenyon
1500 K Street, NW
Suite 700
Washington, DC 20005

METHOD AND APPARATUS FOR CONTROLLING ACCESS TO SHARED RESOURCES IN AN ENVIRONMENT WITH MULTIPLE LOGICAL PROCESSORS

TECHNICAL FIELD

[0001] The present invention relates to multi-processor core or multi-threaded computer microchips, and more particularly to a method and apparatus for controlling access among multiple processor cores or multiple threads on a common microchip to shared resources.

BACKGROUND OF THE INVENTION

[0002] Advances in technology have permitted logic designs to be packed with increased density into smaller areas of silicon as compared with past integrated circuit devices. Moreover, as process technologies have shrunk feature sizes and the sophistication of chip design has increased, chip configurations have evolved from the basic single, central processor model to include two or more processor cores. Additionally, multi-threaded processors are becoming increasingly common.

[0003] Fig. 7 shows an example of a microchip 700 including a plurality of processor cores. Each processor core, 0-N, comprises its own data cache 701, instruction cache 702, and ALU (arithmetic/logical unit) and FPU (floating point unit) execution units 703. Each processor core further has an architecture state 704. The architecture state includes the state of the instruction pointer, the state of all general registers, and other status information such as whether interrupts are enabled or disabled. In the context of execution threads, discussed in more detail below, the architecture state may thus be defined as all the information that is required to resume execution of an interrupted thread. A plurality of processor cores as shown in Fig. 7 typically share certain chip resources. For example, cores 0-N share cache 706, front side bus 707, and control logic 708 via multiplexer 705.

[0004] Fig. 8 shows an example of a microchip 800 including a plurality of execution threads, 0-N. Each thread corresponds to an architecture state. In contrast to the multi-core configuration of Fig. 7, the threads share a common data cache 801, instruction cache 802, and ALU and FPU execution units 803. The threads also share a front side bus 807. A thread identifier register 808 determines, via control inputs to multiplexer 805, which thread is currently executing.

[0005] To operating system software, chips 700 and 800 are functionally the same. That is, the operating system does not differentiate between a core and a thread; whether an operating system request for some processor service is being acted upon by a core or by a thread is transparent to the operating system.

[0006] Accordingly, as used herein, the term "logical processor" refers to either a processor core as shown in Fig. 7, or a thread as shown in Fig. 8.

[0007] The need to integrate the operations of multiple logical processors on a single silicon chip presents a range of challenges. One aspect of these challenges is the need to efficiently manage the shared resources, identified above, used by the respective types of logical processors. Shared resources are defined generally herein as those assets which are available to the different types of logical processors on the same silicon chip.

[0008] It should be understood by "shared" that while more than one logical processor may use the same resource, they generally should not do so concurrently. Thus, more than one logical processor may need a given resource at the same time, but only one logical processor can use it, while the others must wait their turn. The demand among a plurality of logical processors for the same resources requires control schemes to be implemented for managing possible access conflicts.

[0009] In the prior art, such control schemes have typically entailed the use of "semaphores." A semaphore, in the foregoing context, is a data field, usually in a register, that contains information signaling that a particular logical processor of a plurality of logical processors has exclusive use of a shared resource. A logical processor with exclusive use of a resource may be said to have a "lock" on the resource.

[0010] In one known semaphore mechanism, each shared resource has its own guarding semaphore. Such a mechanism suffers from the inherent disadvantage that, if a logical processor requires multiple shared resources, it must serially determine, via each guarding semaphore, the availability of each of the required resources to attempt to obtain a lock on all of the resources. The availability of the required resources may change while the logical processor is attempting to make this determination. If the logical processor cannot obtain a lock on all the resources it needs, it must release the locks on the resources that it was able to obtain. This can lead to a counterproductive condition known as "thrashing," wherein a number of logical processors are repeatedly contending for resources, but no single logical processor is able to obtain all the resources it needs and therefore no forward progress is made.

[0011] In another known semaphore mechanism, a global semaphore is used which locks all available resources for the exclusive use of the logical processor that is able to obtain the lock. However, this approach is inefficient in that all logical processors do not necessarily have overlapping resource needs. Therefore, it could be possible for a plurality of logical processors to use some resources in parallel. A global semaphore lock obviates this possibility.

[0012] Typically, once a logical processor with a semaphore lock on a resource no longer needs the resource, it releases the lock so that other logical processors can use it. However, another undesirable condition that can occur in the use of semaphores is that a logical processor that has obtained a lock may "go bad"; i.e., suffer some kind of hardware or software failure that causes it to retain the lock for an indefinite period of time. This condition, sometimes referred to as "livelock," can also arrest all forward progress among other logical processors. One aspect of known semaphore mechanisms that makes such a problem difficult to resolve is that only the locking logical processor can release the lock.

[0013] An approach is needed to address the concerns noted in the foregoing.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] Fig. 1 is a block diagram of a logic device with two logical processors and access control software for controlling access by the two logical processors to shared resources of the device;

[0015] Fig. 2 is a flow diagram illustrating one possible embodiment of a process implemented by the access control software;

[0016] Fig. 3 shows details of a semaphore register and a resource descriptor according to an embodiment of the invention;

[0017] Fig. 4 shows more examples of the resource descriptor;

[0018] Fig. 5 is a flow diagram illustrating one possible embodiment of a resource descriptor update routine according to the invention;

[0019] Fig. 6 is a flow diagram illustrating an embodiment of a routine for releasing a semaphore lock and reserved resources of a failing logical processor;

[0020] Fig. 7 is a block diagram of a multi-core microchip; and

[0021] Fig. 8 is a block diagram of a multi-threaded microchip.

DETAILED DESCRIPTION

[0022] Embodiments of the present invention as described hereinafter overcome the above-described drawbacks in the prior art. According to the embodiments, access by a plurality of logical processors to shared resources in a logic device may be controlled with a semaphore and a resource descriptor. A first logical processor may update the semaphore in order to obtain exclusive access to the resource descriptor (referred to hereinafter as "obtaining the semaphore lock"). The resource descriptor describes a usage allocation of the shared resources: i.e., it identifies those resources, if any, that are reserved for exclusive use by a particular logical processor, and those resources, if any, that are available.

[0023] Once exclusive access to the resource descriptor is obtained, the first logical processor may query the resource descriptor to determine whether the resource or resources it needs are available. If the resource or resources are available, the first logical processor may update the resource descriptor to reserve the resource or resources for its exclusive use, and then release its exclusive access to the resource descriptor by releasing the semaphore lock. The first logical processor may then freely use the resource or resources it reserved via the resource descriptor.

[0024] Meanwhile, since the first logical processor now no longer has exclusive access to the resource descriptor, a second logical processor may reserve the resource descriptor for its exclusive use by obtaining the semaphore lock. Like the first logical processor, the second logical processor may query the resource descriptor to determine whether the resource or resources it needs are available. If the resource or resources are available, the second logical processor may update the resource descriptor to reserve the resource or resources for its exclusive use, and then release its exclusive access to the resource descriptor by releasing the semaphore lock. The second logical processor may then freely use the resource or resources it reserved via the resource descriptor.

[0025] Similarly, a third, or fourth, or N-th logical processor in a device with N logical processors may then attempt to reserve the resource or resources it needs by the above-described process.

[0026] It may be appreciated that the process avoids the problem of inefficient usage of shared resources that occurs in a mechanism which uses a global semaphore as described above in the background discussion. Instead, shared resources are used efficiently since, if the resource or resources they need are respectively available, a plurality of logical processors may use the resources concurrently.

[0027] Also, the problem of "thrashing" is eliminated since, rather than having to serially obtain locks on semaphores for individual resources, a logical processor can instead obtain a lock via the semaphore to the resource descriptor. Then, the logical processor can determine whether all the resources it needs are available, and if so, reserve all of them, by querying and updating a single data field, i.e., the resource descriptor.

[0028] Fig. 1 is a block diagram illustrating elements in one possible embodiment of the invention. Fig. 1 shows a logic device 100 comprising integrated circuits and software. The device 100 could be, for example, a silicon microchip embodying semiconductor circuits.

[0029] In the example of Fig. 1, the device 100 includes two logical processors 101 and 102.

[0030] Logical processors 101 and 102 may each use shared resources 103. Accordingly, logical processors 101 and 102 may each execute access control software 104 for controlling access to the shared resources between the logical processors. In one embodiment, the access control software may be included in a firmware layer for interfacing between an operating system and chip hardware.

[0031] Logical processors 101 and 102 may communicate with semaphore 105 and resource descriptor 106. Semaphore 105 allows either logical processor 101 or 102 (but not both) to reserve exclusive access to resource descriptor 106. Resource descriptor 106 includes resource identifiers and information about resource availability, as described in more detail below. Semaphore 105 and resource descriptor 106 may both be registers, for example.

[0032] Fig. 2 illustrates a process flow for controlling access between logical processors 101 and 102 to shared resources 103 according to an embodiment of the invention. The flow represents operations by a single logical processor: i.e., it could describe operations by either logical processor 101 or logical processor 102, implemented by executing instructions of access control software 104. For purposes of illustration, however, the following discussion assumes that the process is performed by logical processor 101.

[0033] As shown in ellipse 200, the process may be initiated when code executed by logical processor 101 requires a resource or resources of shared resources 103. When this happens, logical processor 101 may attempt to obtain a semaphore lock on the resource descriptor 106 by updating semaphore 105, as shown in block 201. To obtain the semaphore lock, the logical processor may call a semaphore lock routine of the access control software and pass it the calling logical processor's identifier (i.e., logical processor 101) along with a lock value (e.g. a bit with a "1" value to set the lock) as parameters. The semaphore lock routine may write the logical processor's identifier and lock value into the semaphore register.

[0034] Whether logical processor 101 is able to obtain the semaphore lock may depend upon whether another logical processor (say, 102) already has the semaphore lock. If logical processor 102 already has the lock, logical processor 101 cannot obtain it. Thus, as shown in block 202, after attempting to obtain the lock, logical processor 101 may check to determine whether the lock was successfully obtained. If not, logical processor 101 may loop back to block 201 and try again until it successfully obtains a lock.

[0035] If logical processor 101 successfully obtains a lock, it has exclusive access to resource descriptor 106 (i.e., logical processor 102 is not permitted to change the contents of resource descriptor 106 while logical processor 101 has the lock). Accordingly, as shown in block 203, logical processor 101 may then determine which resource(s) it needs and generate resource reservation data, identifying the needed resource(s), to be applied to the resource descriptor. The resource reservation data, if successfully applied to the resource descriptor, reserves exclusive use of the identified resource(s). The resource reservation data may, for example, be in the form of a bitmap corresponding to the resource descriptor register (Fig. 3).

[0036] Next, as shown in block 204, logical processor 101 may attempt to apply the resource reservation data to the resource descriptor. To attempt to apply the reservation data, logical processor 101 may call a resource descriptor update routine of the access control software and pass it the calling logical processor's identifier and the reservation data identifying the needed resource(s). Whether logical processor 101 is able to successfully apply the resource reservation data to the resource descriptor may depend upon whether logical processor 102 already has reserved a resource or resources that logical processor 101 needs.

[0037] Thus, as shown in block 205, after attempting to reserve the resource(s) it needs, logical processor 101 may check to determine whether the attempt was successful.

[0038] Whether the attempt to obtain all necessary resources (block 205) is successful or unsuccessful, logical processor 101 may release the semaphore lock, as shown in blocks 206 and 208, thereby allowing logical processor 102 a turn at attempting to reserve any resource(s) it may need. To release the semaphore lock, the logical processor may call a semaphore lock release routine of the access control software and pass it the calling logical processor's identifier (logical processor 101) and lock value (e.g. a bit with a "0" value to reset or unlock the lock) as parameters.

[0039] If the attempt was not successful, logical processor 101 may, in effect, start over, releasing the semaphore lock as shown in block 208 and "getting back in line," by returning to block 201.

[0040] If the attempt was successful, on the other hand, logical processor 101 may freely use the reserved resource(s), as shown in block 207.

[0041] When logical processor 101 is finished using its reserved resources, the resource descriptor needs to be updated accordingly, so that logical processor 102 can use the resources that logical processor 101 was using if it needs to. Thus, logical processor 101 needs to obtain exclusive access to the resource descriptor, by obtaining a semaphore lock. Therefore, as shown in blocks 209 and 210, logical processor 101 may attempt to obtain the semaphore lock as described above with reference to blocks 201 and 202.

[0042] When the semaphore lock is obtained, logical processor 101 may release the resource(s) it reserved as shown in block 211. To release the resource(s), logical processor 101 may call a resource descriptor release routine similar to the resource descriptor update routine mentioned above. Of course, since logical processor 101 has the resource(s) reserved, there need not be any test, equivalent to block 205, as to whether logical processor 101 can release the resource(s).

[0043] As shown in block 212, logical processor 101 may then release the semaphore lock, allowing logical processor 102 to obtain the lock, if needed.

[0044] The foregoing description illustrates in detail how the invention allows for parallel use of shared resources by logical processors 101 and 102. Having once obtained the semaphore lock, whether or not logical processor 101 is able to reserve the resources it needs, it then releases the semaphore lock, allowing logical processor 102 to obtain the semaphore lock and attempt to reserve the resources that it needs. Thus, for example, if logical processor 101 needs resources A, B, and C, while logical processor 102 needs resources D, E, and F, both logical processors can reserve and use their respective resources concurrently.

[0045] Further, "thrashing" is eliminated because once a logical processor has the semaphore lock, no other logical processor can reserve resources. Thus, the availability of resources does not change while the logical processor holding the semaphore lock is attempting to reserve resources. And, the logical processor holding the semaphore lock can determine the availability of, and reserve the resources it needs by simply reading and updating the resource descriptor.

[0046] The foregoing process was described with reference to only two logical processors merely by way of example, in order to illustrate the principles of the invention. The device 100 could comprise, and the process would work, of course, with a number of logical processors greater than two.

[0047] Fig. 3 shows examples of possible embodiments of semaphore 105 and resource descriptor 106. Semaphore 105 could be an N+1-bit register, where N is the number of bits needed to encode the number of logical processors in the device 100. The low-order (0-th) bit position could be used for a lock bit as described above, while the 1st through N-th bits could contain a logical processor identifier. The register could be read/write (RW) shared.

[0048] Resource descriptor 106 could be a register formatted into a plurality of fields each associating a resource with a logical processor identifier (LPID) and a status identifier such as a lock bit. Each field associating a resource with a LPID and a status identifier could identify the corresponding resource based on the field's position, i.e., order, within the register. Each field could be N+1 bits long, with the low-order bit acting as the lock bit and the higher-order bits holding the LPID. Each resource's LPID field would identify which, if any, logical processor had reserved the resource, and the status identifier would indicate whether the reservation was currently active.

[0049] Fig. 4 shows three specific examples of resource descriptors 106.1, 106.2 and 106.3 for N=1, 2, 3, respectively.

[0050] As described above, a logical processor which successfully obtains a semaphore lock on the resource descriptor may generate resource reservation data in the form of a bitmap to be applied, if possible, to the resource descriptor to reserve exclusive use of the needed resource(s). Table 1, below, shows one possible format for the bitmap:

Table 1

Bit Location:	M	...	6	5	4	3	2	1	0
Resource:	Res M	...	Res 6	Res 5	Res 4	Res 3	Res 2	Res 1	Res 0

(where M is the number of shared resources)

[0051] Thus, for example, in a device 100 with 10 shared resources, a logical processor needing resources 2, 5 and 8 might generate a bitmap as follows: 0100100100.

[0052] As further described above, to attempt to apply the bitmap, the logical processor may call a resource descriptor update routine of the access control software and pass it the calling logical processor's identifier (LPID) and the bitmap identifying the needed resource(s). The resource descriptor update routine may generate a new resource descriptor based on the bitmap, calling LPID and the current resource descriptor, and write the new resource descriptor into the resource descriptor register. Figure 5 shows a flow diagram for one possible embodiment of such a resource descriptor update routine.

[0053] As shown in ellipse 500, the routine may begin once the logical processor obtains the semaphore lock. Block 501 represents the establishment of an index, j, for stepping through the resources in the resource descriptor.

[0054] Block 502 shows testing a value of a bit in the bitmap, where "Resource_Bit_Map" is an arbitrary mnemonic assigned to the bitmap. If a bit has a "1" value, meaning that the logical processor needs to reserve the corresponding resource, the routine may progress to block 503. Otherwise, the routine may progress to block 505. Block 505 represents simply copying the current j-th resource descriptor field to the new j-th resource descriptor field.

[0055] Block 503 represents reading the current resource descriptor to test whether the resource required by the bitmap is already reserved. "Resource (j).LOCK" is an arbitrary mnemonic assigned to the current lock bit of the j-th resource. If the lock bit has a value of "0", the resource is available. Accordingly, the routine may branch to block 504. If, on the other hand, the requested resource is reserved, the routine may exit.

[0056] Block 504 represents generating a new field for the j-th resource of the new resource descriptor, where the new field is different from the current field. "New_Resource (j).LOCK" is an arbitrary mnemonic assigned to the lock bit of the new field. The update routine may set the lock bit to "1". "New_Resource (j).LPID" is an arbitrary mnemonic assigned to the LPID field of the new field. The routine may write the calling LPID to the new LPID field.

[0057] The routine may then progress to block 506 and the next loop iteration. If all the bits in the bitmap have been processed, the routine may then write the new resource descriptor to the resource descriptor register, as shown in block 507.

[0058] As discussed earlier, one of the problems that can occur in the use of semaphores is that a logical processor may fail while it has a lock on needed resources. This situation can be difficult to remedy because read/write access to the semaphore register is hardware-controlled, and only the locking logical processor is authorized to release the lock. In known systems, it is the logical processor hardware which supplies its LPID to the hardware which controls access to the semaphore in order to obtain the lock.

[0059] By contrast, in embodiments of the present invention, the access control software supplies the LPID to the hardware which controls access to the semaphore . That is, a logical processor seeking to obtain a semaphore lock calls a semaphore lock routine of the access control software and passes the routine its LPID and a lock value. Similarly, a logical processor seeking to release a lock calls a semaphore lock release routine of the access control software and passes the routine its LPID and a lock value. It is the semaphore lock routine or semaphore lock release routine that passes the LPID and lock value to the semaphore access control hardware.

[0060] Thus, because the LPID is being supplied by software rather than hardware, it is possible for a different logical processor to, in effect, masquerade as the failing logical processor in order to release the lock of the failing logical processor on the semaphore. That is, when a failing logical processor is detected and the failing logical processor has the semaphore lock, another logical processor can call the semaphore lock release routine and pass the routine the LPID of the failing logical processor. Since the "authorized" LPID has now been supplied, the semaphore can release the lock.

[0061] Embodiments are illustrated in Fig. 6. When a failing logical processor (LP) is detected (block 600), it may be determined whether the failing logical processor can fix itself, as shown in block 601. If so, the process may return directly (i.e., the resources locked by the failing logical processor will be freed up in the normal course of events).

[0062] If not, block 602 may be executed to determine whether the failing logical processor has the semaphore lock. If the failing logical processor has the semaphore lock, another logical processor may call the semaphore lock release routine and pass it the LPID of the failing logical processor, as shown in block 603. The semaphore lock release routine may pass the LPID of the failing logical processor to the semaphore access control hardware, along with a lock value of "0", thereby releasing the semaphore lock.

[0063] The process may then continue to block 604 and 605, where after obtaining the semaphore lock, it may be determined whether the failing logical processor has any resource(s) locked. If it does, another logical processor may call the resource descriptor release routine and pass it the LPID of the failing logical processor along with a bitmap setting the respective lock bits of the reserved resources to "0", as shown in block 606. All resources reserved by the failing logical processor will then have been released. The semaphore lock may then be released as shown in block 607, and the process may return.

[0064] Hardware on the device 100 may be configured to support the above-described software-implemented processes. For example, according to embodiments each logical processor on the device may include a register containing its LPID, which is readable by the access control software 104 to enable the access control software to identify which logical processor it is executing on. Further, logical processors on the device could have unrestricted read access to the semaphore and resource descriptor registers.

[0065] Additionally, upon an attempted write to the semaphore register 105 by a logical processor, the semaphore access control hardware could be configured to check the status of the lock bit of the semaphore register. If the lock bit is set, the hardware could compare the incoming LPID, passed to the semaphore access control hardware by the access control software, of the logical processor attempting the write with the current LPID in the semaphore register, and only allow the write if the two LPIDs were the same. As noted above, because the LPID is supplied by the access control software, there is the possibility of releasing a lock of a failing logical processor by supplying the failing logical processor's LPID to the hardware.

[0066] On the other hand, if the lock bit is not set, the semaphore access control hardware could allow a write to the semaphore register from any logical processor.

[0067] Further, if two or more logical processors were attempting to write to the semaphore register at the same time, device hardware could be configured to give priority to a write which was attempting to release the semaphore lock over writes attempting to obtain the lock. This should be done because the assumption is made in software that the write to release has succeeded.

[0068] Access control software comprising computer-executable instructions according to embodiments of the present invention may be stored and transported on a computer-usable medium such as diskette, magnetic tape, disk or CD-ROM. The instructions may be downloaded to another storage medium such as a ROM or RAM on device 100, from which they may be fetched and executed by logical processors of device 100 to effect the advantageous features of the invention.

[0069] Several embodiments of the present invention are specifically illustrated and described herein. However, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.